Journal of Computer Science 3 (8): 666-672, 2007
ISSN 1549-3636

A Novel Service Mobility Architecture for SIP Environments

Thierno Bah, Samuel Pierre, Roch Glitho
Department of Computer Engineering, École Polytechnique de Montréal, C.P. 6079, Station succ. Centre-ville, Montréal, Québec, Canada H3C 3A7

Abstract: Lately, the proliferation of small portable devices has driven the introduction of a new mobility concept called service mobility. In order to support service mobility, we introduce a new architecture implementing mechanisms to access end user's personalized services regardless of his physical location. The proposed architecture leverages on mobile agent technology to transport and move services to the end user's registered terminals and the Session Initiation Protocol (SIP) for user location signaling. A decentralized approach purely based on mobile agents is compared with a centralized approach. The performance measurement results show that mobile agents are worth being used for realizing service mobility but in specific conditions.

Keywords: mobile agents, service mobility, SIP

INTRODUCTION AND BACKGROUND

In current generations of mobile networks (1G, 2G and 2.5G), terminal and user mobility are well supported. However, those forms of mobility cannot guaranty a global mobility with the proliferation of handheld devices and the heterogeneity that will characterize the next generation of mobile networks. New forms of mobility have therefore emerged. Service mobility is intended for third (3G) and beyond (4G) generations of mobile networks. This new form of mobility is defined as the ability for the user to transparently access his personalized services from any terminal with the same profile while moving.

The main motivation for supporting service mobility is the ability to follow end users while providing them with their services with the same look and feel as they move. To do so, a distributed communication paradigm is needed to update the services and a signaling protocol is also needed to track the user's location and provide multimedia services. The mobile agent paradigm can be used in an elegant way to support service mobility in a mobile environment [1,2]. Mobile agents are entities that can start execution in a node, suspend execution, and move to another node to resume execution. This new paradigm presents many advantages for the realization of service mobility. Those advantages are, but are not limited to, autonomy, mobility and persistency. It is also claimed that mobile agents reduce bandwidth consumption, reduce processing delays, easily support user personalization and are not affected by network conditions. More information on mobile agents can be found in [3] for the interested reader. The Session Initiation Protocol (SIP) is a signaling protocol for Internet telephony that has been officially chosen as the standard signaling protocol for third generation networks [4]. It allows the introduction of new types of multimedia services integrating voice, data, and video. We truly believe that SIP will be a major enabler for service creation and provisioning in next generation networks.

In this paper, we propose a new service mobility architecture for roaming users that frequently change working terminals. This new architecture is based on Emako et al.'s generic service architecture [5]. An overview of the latest work on service mobility architectures is presented next. Then we present our architecture and the two schemes we have implemented for the service update. We finally evaluate the performance of the proposed schemes and discuss the strengths and weaknesses of each one.

Many service architectures have been developed lately for next generation networks [6]. Some rely on legacy tools such as Intelligent Networks (IN); others rely on new paradigms like mobile agents. In this overview, we focus on the few service architectures that address a form of service mobility. The SOMA infrastructure [7] defines a framework called User Virtual Environment (UVE) that is implemented by a

Corresponding Author: Department of Computer Engineering, École Polytechnique de Montréal, C.P. 6079, Station succ. Centre-ville, Montréal, Québec, Canada H3C 3A7

mobile agent and aims to follow end users when they move. Another relevant architecture targeting well-known Internet services (HTTP, FTP…) is NetChaser [8].

Service mobility is defined as the ability for the user to transparently access his personalized services from any terminal with the same profile while moving. To do so, any service architecture supporting service mobility must address user location tracking, transparent updating of services and personalization of service data. Service mobility can be realized in an elegant way by implementing a mobile agent to transport a particular service and follow end users when they move. In our work, we have defined a unique mobile agent acting as a folder and transporting all the subscribed services. This approach should be more suitable for better management and to avoid performance and scalability problems.

Another major contribution we would make with this work is to provide empirical measurements to analyze the performance of the proposed mobile agent based architecture. In fact, the related work does not provide any empirical measurements and/or an analysis of the performance of the proposed architecture. However, since performance can be a major issue in such a mobile environment with limited portable devices and scarce resources, we have decided to devote a large part of our work on experimentations and performance measurements.

Service Mobility Architecture: We rely on the architecture proposed by Emako et al. [5] to build an architecture supporting service mobility. The key components of the architecture are the Mobile Service Agent (MSA) that acts as a folder and transports the subscribed services; the Service Management Unit (SMU) is the entity that manages the users' subscribed services and the associated profile through the MSAs; and finally the Service Creation Unit (SCU) offers the appropriate service creation environment.

Requirements: To support service mobility, any architecture must provide some tools for the management of the user's subscribed services. To do so, some requirements must be respected. Those requirements are:

- Transparency: the mobility management mechanism must be transparent to the end user;
- User location tracking: the user must be tracked when changing locations.
- Service updating: the service must be updated with the latest version in all the terminals being used;
- Data personalization: the service data must be personalized with the latest user profile in all the terminals being used;
- Persistency: a connectionless mode of operation should be supported.

A function for adapting the service content to the terminal capabilities could also be required in a heterogeneous environment. This field is actively researched, and the interested reader is referred to [9]; however, this is out of the scope of this paper.

Software components: Our architecture relies on SIP [10] as the underlying protocol for signaling. We implement new schemes and new functions in the MSA and SMU to support service mobility. Service updating and data personalization functions are added in the MSA's coordinator (logic). A SIP interface for user location tracking and an MSA manager are also added to the SMU. The mobility can be achieved by using the « REGISTER » message of the SIP signaling protocol [11]. When receiving such a message, the SIP server then alerts the SMU that keeps track of the user's whereabouts. Mobile agents follow end users to realize the updating and personalization functions. Howeverm these functions in the coordinator of the MSA can only be initiated by an entity with the appropriate credentials for security reasons. This control can be simply performed with an authentication certificate. More materiel on the security issues with the mobile agent paradigm can be found in the following reference for the interested readers [12].

The components of the architecture are depicted in Figure 1. The illustrations in the remaining pages are limited to 3 terminals for practical reasons; however, the architecture is scalable enough to support as many terminals as the user desires, as it will be demonstrated with the performance measurements. The SMU is made of two main modules: the SMUInterfaceAgent and the MSAManagerAgent. The SMUInterfaceAgent is a stationary agent. It interfaces with the SCU and makes the newly created services available to the user. It also interfaces with the SIP location server and receives a notification whenever a user logs into a new terminal. The location (address) of the user is part of the notification message that is sent to the server. The SMUInterfaceAgent forwards the address to the MSAManagerAgent. The MSAManagerAgent is also a stationary agent. It creates, then tracks and manages all the MSAs through proxies.

The modules that make the MSA are depicted in Figure 2. As shown in the figure, the support of service mobility is mainly realized with two new operations (i.e., "Updating", and "Personalization") to the set of native operations the MSA can carry ("clone" for example).

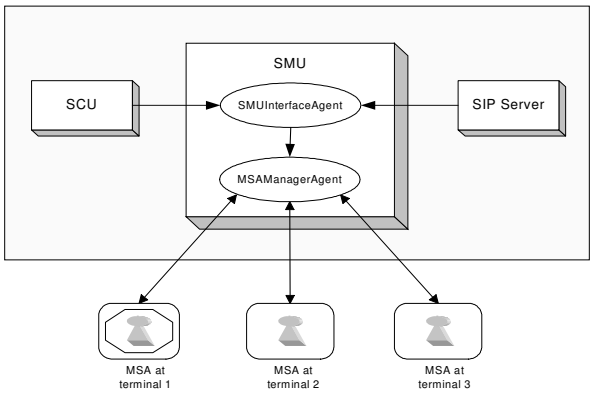

Fig. 1: The main components of the architecture

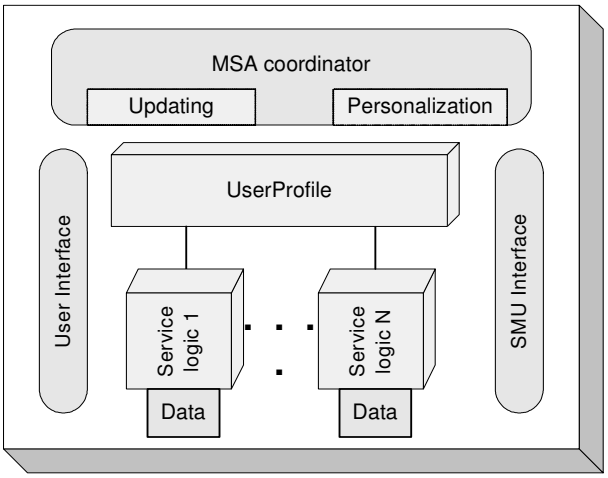

Fig. 2: Software architecture of the Mobile Service Agent (MSA)

Dynamic updating of the MSA: In order to minimize the downtime due to the update of the services, the MSAs have to be updated dynamically. This update can be done by substituting, adding, and/or deleting Java classes using dynamic class loading, which is a powerful tool of the Java virtual machine. This underlying mechanism provides the Java platforms with the ability to install software components at runtime. Liang and Bracha [13] describe this approach. Running software is made of multiple classes that are loaded in the running Java virtual machine. Updating this software consists in reloading the new versions of the changed classes in the running virtual machine. The principle is simple, but the implementation is not trivial. In fact, there are many live objects that are instances of the class to be reloaded. Since these objects are live entities, they cannot be updated to conform to the new version of the modified class even if the class has been reloaded. In fact, a potential problem arises when the application is executing a method that belongs to an instance of a class that needs to be reloaded. In this case, the application is forced to finish executing the method before the new version of the class can be loaded. If the new version does not define the method or if it has a new version of the method, the old version is executed anyways. However, those problems called the schema evolution problem can be bypassed easily by adopting a modular approach and organizing the software into different class loaders [14]. The following example will illustrate that fact. A user subscribes to a service by adding an instance of the Service class in his profile. A first approach is to define the Service class as a whole class containing the data of the service and the specific methods for executing the service. Instead, another approach (the one we use here) organizes the service into a modular entity made of two different classes: a Method class containing the methods for executing the service and a Data class containing the specific data of the service. In this way, we only need to reload the Data class if the user modifies the data (preferences) of the service instead of reloading the whole class, therefore avoiding the potential problems associated with reloading an active class.

Service Updating and Personalization Schemes: Two novel schemes are proposed to address service updating and personalization for service mobility. For both schemes, the location of the user and the migration of the first MSA is performed the same way through SIP signaling. As soon as the user registers at a new terminal sending a "REGISTER" message to the SIP location server, the server sends the terminal address to the SMU. The SMU creates an MSA containing the subscribed services with the customized data and moves it toward the new terminal. The updating approach is however different, depending on the used scheme. The first scheme uses a hybrid approach that is centralized. In fact, mobile agents are used to follow mobile users and RMI is used to synchronously update the agents. The updating is initiated by the central entity i.e. SMU that remotely invokes the "updating" and/or the "personalization" function of the MSA using RMI calls with the services and/or user data profile as parameters of the call. This approach is however optimized by initiating the RMI calls in parallel for each terminal, without having to wait for the synchronous process to finish on a given terminal before initiating it on another terminal. The second scheme uses a pure agent based approach that is decentralized. In fact, in this approach, the updating process is delegated to an updating agent that is created at the SMU unit. That updating agent asynchronously updates the MSAs after touring all the terminals where he calls the "updating" and/or the "personalization"

function locally. The major benefit of delegating the updating task to a mobile agent is that the updating is done locally and asynchronously in a decentralized fashion. Furthermore, this approach offers a high level of persistency since if a terminal is disconnected from the network, the updating agent waits until the terminal becomes available to accomplish its task.
A full scenario involving 3 terminals is depicted in Figure 3.

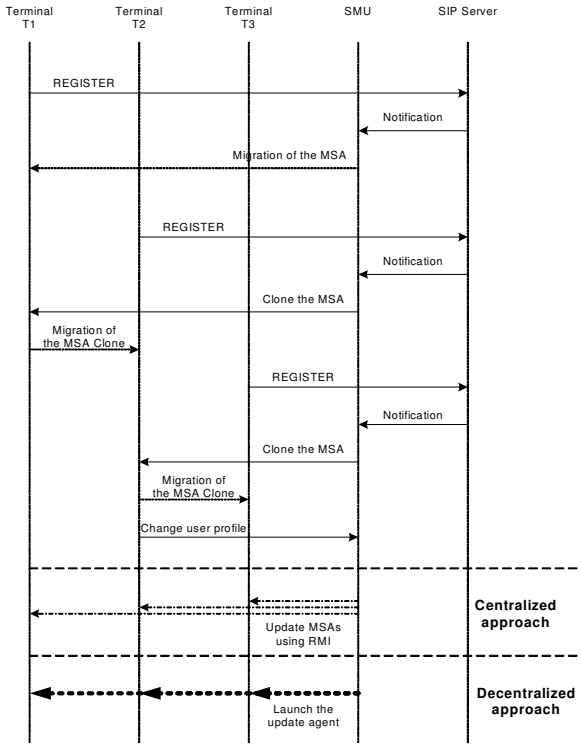

Fig. 3: Sequence diagram illustrating the two schemes

The scenario goes as follows:

- The user registers at terminal T1. The SIP location server sends the terminal address to the SMU. The SMU creates an MSA containing the subscribed services and customized data and sends it to terminal T1.
- The user now registers at T2 while still registered at T1. The SIP location server informs the SMU that the user has logged in at terminal T2. The SMU tells the MSA located at T1 to clone itself. A new MSA is created and migrated to T2.
- The user now registers at T3, in addition to T1 and T2. The SIP location server informs the SMU that the user has logged in at T3. The SMU tells the MSA at T2 to clone itself and initiates the migration of the newly created MSA at T3.
- Now the user makes changes to his service profile at T2 (registers to a new service and/or updates the version of a registered service and/or changes the preferences data of a registered service…). The SMU is then notified. Two schemes are now possible depending on the approach that is used:

Centralized approach: the SMU asks the MSAs located at T1, T2 and T3 to proceed with the updates by remotely initiating a call to their "updating" and/or "personalization" function using RMI. The calls are initiated in parallel to optimize the process. The changes to be made are transmitted as a parameter of the calls. When it is a change to the preferences data of a registered service, the request is only sent to T1 and T3.

Decentralized approach: the SMU creates an updating agent containing the changes and initiates its migration to terminal T1, to terminal T2 if necessary (in the case the user registers to a new service or updates the version of a registered service; it is not relevant to send the updating agent to T2 if it is a change to the preferences data of a registered service) and to the terminal T3 where it updates the MSA locally.

Performance Evaluation :To analyze the performance of the architecture we have proposed, we built a test environment based on the two schemes we proposed earlier. We made measurements of the delay and network load involved in the updating process for each scheme. For this first contribution, we built our test environment using a static network, where all the workstations were connected to the same segment of a 100 Mbits/second Ethernet bus in the local area network.

Metrics: We used two metrics in our analysis: the delay and the network load.
The delay measures the duration of the updating process. It is influenced by three main factors:

- The size of the service logic: when a new version of the service is available, the service has to be transferred to all the MSAs. The transfer delay depends on the size of the service.
- The size of the service data: when the user makes changes to customized data, the new data has to be transferred to the MSAs on the other terminals. The transfer delay depends on the size of the service data.
- The number of terminals where the user is registered: since each terminal the user is registered in contains an MSA that has to be updated, the updating time depends on the number of involved terminals.

The network load measures the amount of data transferred in the network during the updating process. This metric, like the previous one, is mainly influenced

by the size of the service logic, the size of the service data and the number of terminals.

RESULTS AND DISCUSSION

We measured the delay as a function of the number of terminals. The measurements were obtained from several evaluation sessions conducted for a whole week during the night (when the network is not very loaded) in order not to influence the results. During this period, the local network latency was around 0.4s and the average available bandwidth was around 75 Mbits/second. To get more accurate results, we reported the data from 10 samples. We took the average of 5 samples with a standard deviation less than 5 %. Results are shown for a big service (Figure 4a with a size of 5000 Kbytes), a small service (Figure 4b with a size of 500 Kbytes) and the service data (Figure 4c with a size << 500 Kbytes) for the centralized (C) and decentralized (D) approaches.

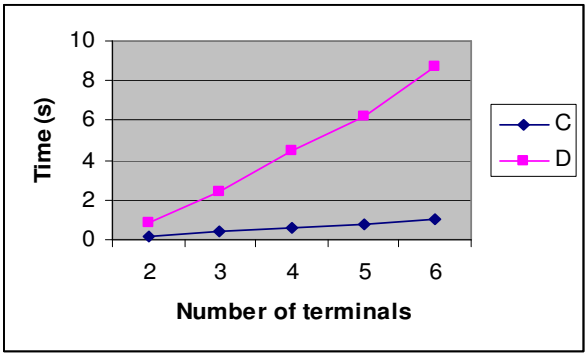

Fig. 4c Delay – Service data

The network load is also plotted as a function of the number of terminals for a big service (Figure 5a with a size of 5000 Kbytes), a small service (Figure 5b with a size of 500 Kbytes) and the service data (Figure 5c with a size << 500 Kbytes).

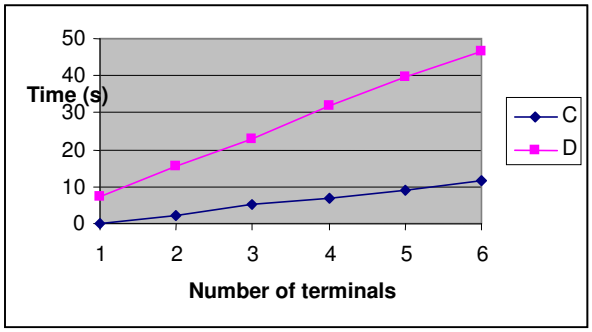

Fig. 4a Delay – Big service

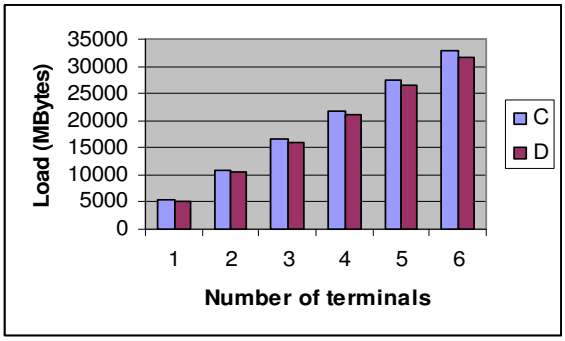

Fig. 5a Network load – Big service

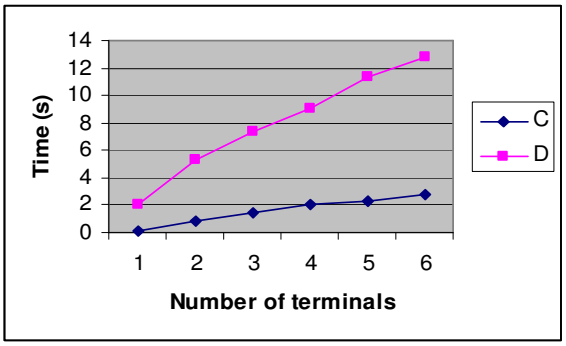

Fig. 4b Delay – Small service

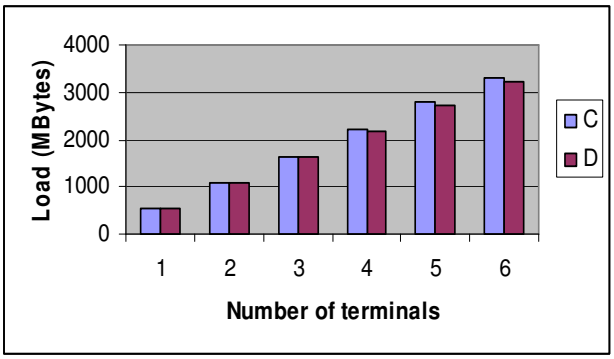

Fig. 4b Network load – Small service

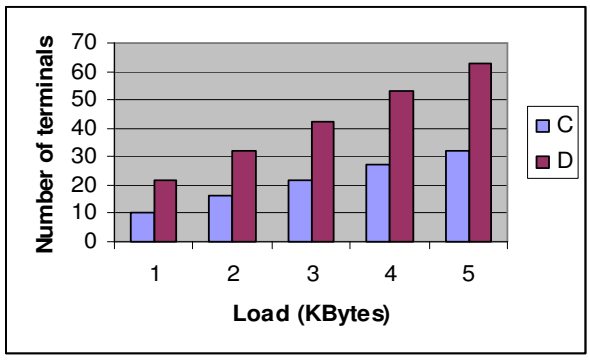

Fig. 4c Network load – Service data

In all cases, the centralized approach has a delay that is much more important than its counterpart as depicted in Figures 4a, b and c. In fact, the decentralized approach involves many steps that require a lot of processing: the creation of the updating agent; the transmission of the service code to the agent; the serialization; the migration; and the un-serialization of the agent at each destination. Furthermore, the updating agent has to migrate several times to reach all the MSAs on all the connected terminals. On the other hand, the centralized approach only involves the transmission of the new service or data to the MSAs through a remote call to the updating method without any further processing. This process is furthermore optimized by initiating the remote calls in parallel specifically to reduce delay.

When it comes to the network load, the decentralized approach outperforms the centralized one in specific cases. In fact, the decentralized approach creates less traffic than the centralized one when heavy data is involved in the transfer. It creates less network traffic for a service transfer (heavy data) as depicted in Figures 5a and 5b while creating much more traffic for a service preference data transfer (small data) as depicted in Figure 5c. This can be explained by the fact that the agent only transports its data state, which consists of all the non-transient instance variables of the agent class, and does not transport any transient data as it is the case with RMI in the hybrid approach. Also, intermediate results are not transmitted when using mobile agents in the decentralized approach. However, those benefits are apparent only for a data size that is important enough to counterbalance the overhead associated with the creation of the mobile agent. Therefore, to be efficient when using a mobile agent based approach, the load that has to be transferred must reach a specific data size threshold. This threshold is around 500 Mbytes in our experience. It is however dependant on the nature of transfer.

CONCLUSION

In this paper, we proposed a new architecture to support service mobility. A short critical overview of the existing architectures was done, requirements were identified and new features and schemes were proposed to meet those requirements. A performance evaluation was done and results analyzed.

In the scope of this work, we used static terminals for practical reasons to realize the experiences. In future work, we will used mobile terminals, running our experiments in a mobile wireless network that is closer to the targeted networks (3G and beyond). The schemes we use are also static. In the future, we shall derive algorithms for deciding which scheme to use, depending on the metrics (delay and/or network load optimization) we want to optimize. In addition we could explore schemes where there is no centralized network entity involved in the updates with cooperating agents. Such schemes could be useful for mobile Ad hoc networks. Other paradigms such as emerging web services could also be used to realize the service mobility.

REFERENCES

1. Vasiu L. and Mahmoud Q. H., 2004. Mobile agents in wireless devices. IEEE Computer, 37 (2): 104 – 105.
2. Yu Y. and Zhang P., 2003. Service mobility in mobile network. Proceedings International Conference on Communication Technology ICCT. 2 (1): 1698 – 1701.
3. Chess D., Harrison C. and Kershenbaum A., 1995. Mobile Agents: Are They a Good Idea? IBM Research Division, T.J. Watson Research Center, Yorktown Heights, New York, available at URL: http://www.cs.dartmouth.edu/?agent/papers/chapter.ps.Z. 16
4. Glitho R., 2000. Advanced Service Architectures for Internet Telephony: A Critical Overview. IEEE Network Magazine, 14 (4): 38-44.
5. Emako B., Glitho R. and Pierre S., 2003. A Mobile Agent based Advanced Service Architecture for Wireless Internet Telephony: Design, Implementation and Evaluation. IEEE Transactions on Computers, 52 (6): 690-705.

6. Glitho R., 2001. Emerging alternatives to today's advanced service architectures for Internet telephony: IN and beyond. Computer Networks, 35 (5): 551-563.
7. Bellavista P, Corradi A. and Stefanelli C., 2001. Mobile Agent Middleware For Mobile Computing. IEEE Computer, 34 (3): 73-81.
8. Stefano D. and Santoro C., 2000. NetChaser: Agent Support for Personal Mobility. IEEE Internet Computing Magazine, 4 (2): 74-79.
9. Timmerer C. and Hellwagner H., 2005. Interoperable adaptive multimedia communication. IEEE Multimedia, 12 (1): 74 – 79.
10. Schulzrinne H. and Rosenberg J., 2000. The Session Initiation Protocol: Internet Centric Signaling. IEEE Communications Magazine, 38 (10): 134-141.
11. Lou D., Jiang D., Yeap T. and O'Brian W., 2005. Personalized service mobility and security in SIP-based communications. 13th IEEE International Conference on Communications, 1 (1) : 113-117.
12. Benachenhou L. and Pierre S., 2006. Protection of a Mobile Agent with a Reference Clone. Computer Communications Special issue on Dependable Wireless Sensor Networks, 29 (2): 268-278.
13. Liang S. and Bracha G., 1998. Dynamic Class Loading in the Java Virtual Machine. Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, 1 (1) : 36-44
14. Hirschfeld R. and Kawamura K, 2006. Dynamic service adaptation: Experiences with Auto-adaptive and Reconfigurable Systems. Software Practice and Experience, 36 (11) : 1115-1131.